The augmenting path algorithm

Def : given a flow x, an augmenting path is a directed path from s to t in $G(x)$; its residual capacity is the minimum residual capacity of its arcs.

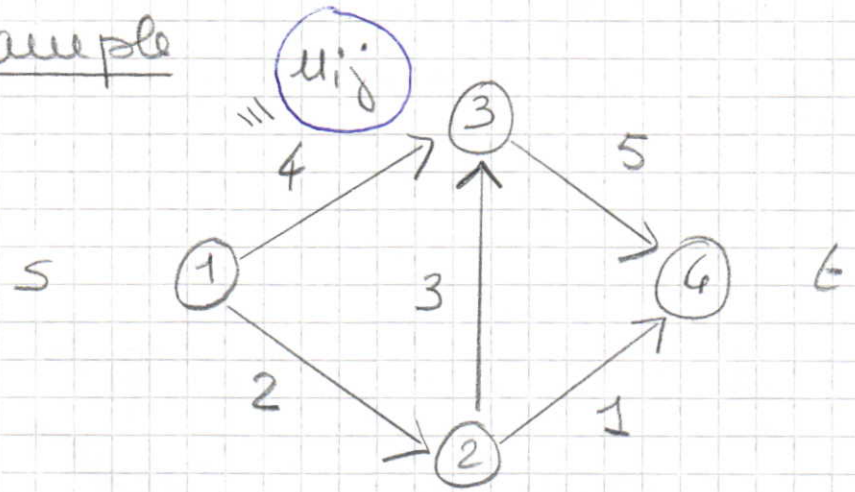
The algorithm : at each iteration finds an augmenting path and augments flow on this (= residual capacity) until there is no augmenting path

Theorem : a flow x^* is a maximum flow if and only if $G(x^*)$ contains no augmenting path ;

Proof : from the max-flow min-cut theorem (course RO)

Theorem (integrality): if the arc capacities are integer, then there exists an integer maximum flow.

example



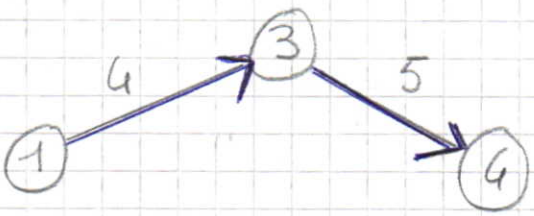
Initial flow

$x = 0$

$v = 0$

1) $G(x) = G$

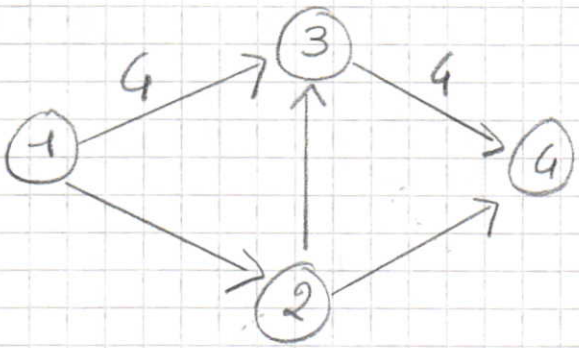
aug. path



$\delta = \min\{4, 5\} =$

4 residual capacity

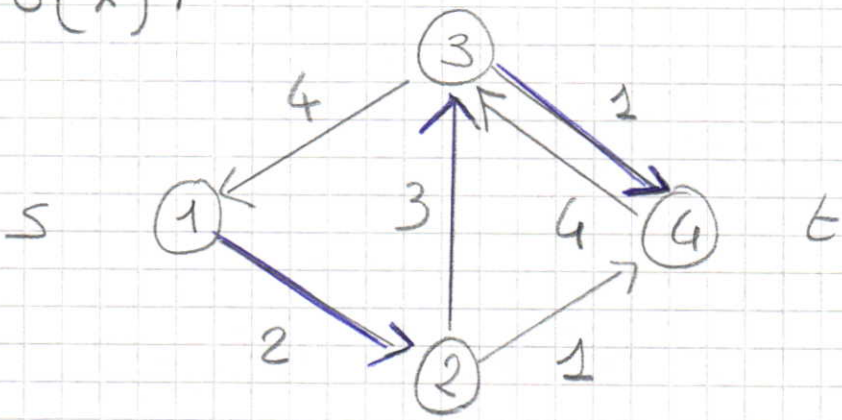
flow x



$v = 4$

NB: an integer flow at each iteration!

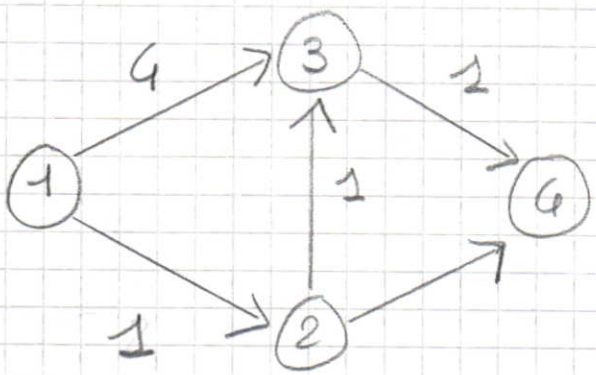
2) $G(x)$:



aug. path : $(1, 2, 3, 4)$

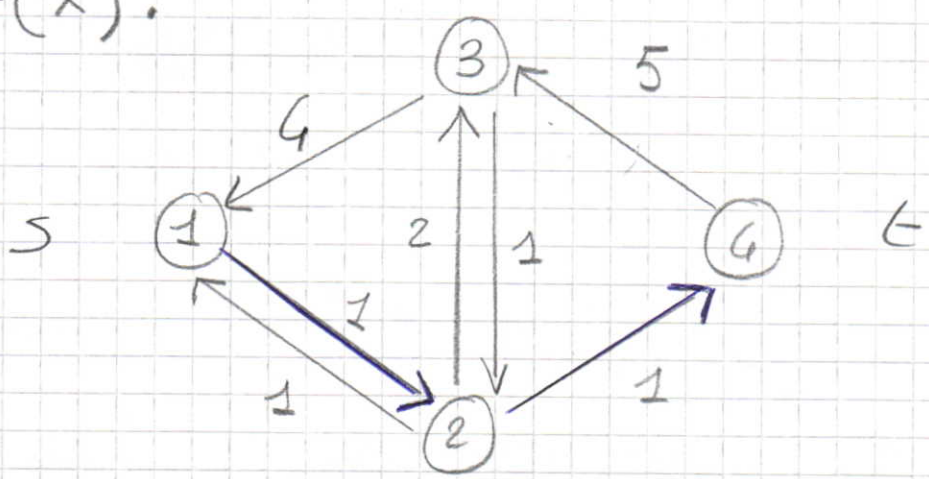
$\delta = \min\{2, 3, 1\} = 1$

flow x :



$v = 4 + \delta = 5$

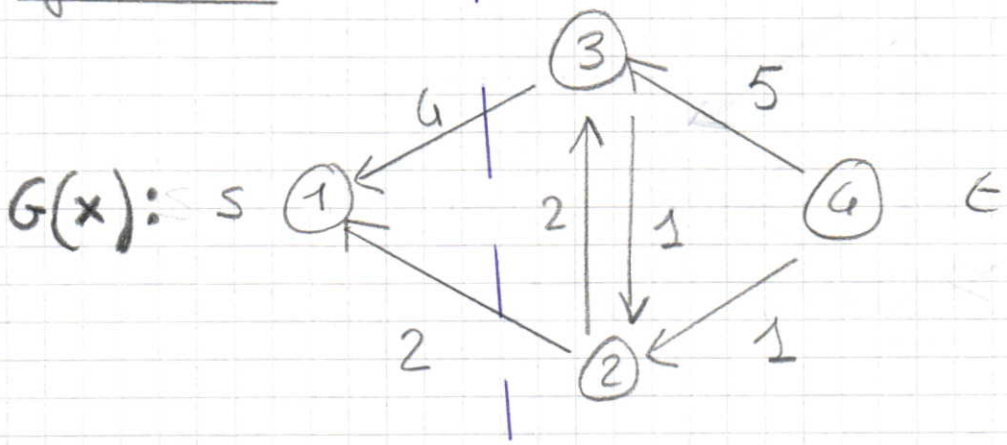
2) $G(x)$:



aug. path : $(1, 2, 4)$

$\delta = \min\{1, 1\} = 1$

flow x :



$v = 5 + 5 = 6$

no augmenting path :

STOP, x is an (integer) maximum flow

A minimum cut : $S = \{1\}$ $\overline{S} = \{2, 3, 4\}$

$$u[S, \overline{S}] = u_{12} + u_{13} = 6 = v$$

Time complexity ($u_{ij} \in \mathbb{Z}^+$) : $O(n \cdot m \cdot U)$, where U maximum arc capacity

Proof :

- capacity of any s-t cut : $O(n \cdot U)$
- time to discover an augmenting path : $O(m)$

□

Drawbacks

1) for large U, the augmenting path algorithm may perform many iterations (see pathological example 6.18);

2) if u_{ij} are irrational, it may not terminate, and converge to a value < maximum flow value;

3) it is "forgetfulness": at each iteration it looks for an augmenting path starting from "scratch" (it erases information that could be useful in next iterations).

How can we overcome some of these drawbacks?

Maximum flow : polynomial algorithms

(Ahuja - Magnanti - Orlin : Chapter 7 (7.1, 7.2, 7.4 (until page 214, "Correctness of the Algorithm" excluded), 7.6)

1) refinements of the generic augmenting path algorithm : use a combinatorial strategy to limit the number of used augmenting paths →

shortest augmenting path algorithm

2) relaxation algorithms which use shortest augmenting paths, but do not send flow from s to t, but along individual arcs →

preflow - push algorithms

↗ powerful theoretically and computationally

In both cases an important concept is

Distance labels

Let $G=(N,A)$ with flow x

$d: N \to \mathbb{Z}^+ \cup \{0\}$ is a distance function

d is valid w.r.t. x if:

i) $d(t)=0$

ii) $d(i) \leq d(j)+1 \quad \forall (i,j)$ in $G(x)$

< validity conditions >

$d(i)$ is said the distance label of i

Property: if d is valid, then $d(i)$ is a lower bound on the cardinality of the shortest path from i to t in $G(x)$, $\forall i \in N$.

Proof:

Let

i $(i_1) \longrightarrow (i_2) \longrightarrow \dots \longrightarrow (i_{k+1})$ t

a path from i to t in $G(x)$ composed of k arcs.

From the validity conditions:

$$d(i_k) \leq d(i_{k+1}) + 1 = d(t) + 1 = 1$$

$$d(i_{k-1}) \leq d(i_k) + 1 \leq 2$$

$$d(i_{k-2}) \leq d(i_{k-1}) + 1 \leq 3$$

$$\vdots$$

$$d(i) = d(i_1) \leq d(i_2) + 1 \leq k$$

Therefore $d(i) \leq$ cardinality of any path from i to t in $G(x)$, and also of the shortest one

□

Consequence: If $d(s) \geq n$, then there is no path from s to t in $G(x)$, i.e. no augmenting path

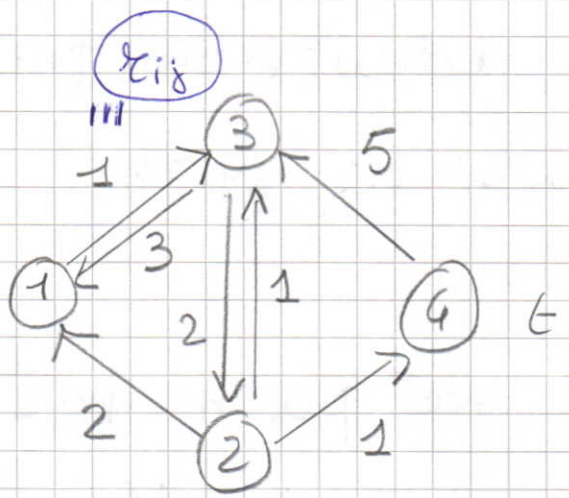
Proof: these paths have $\leq (n-1)$ arcs

Additional notation :

1) $d(i)$ is an exact distance label if $d(i)$ = cardinality of shortest path from i to t in $G(x)$

(computable in $O(m)$ via a backward breadth-first visit of $G(x)$ starting from t)

example



$G(x)$: s ... t

$d = (0, 0, 0, 0)$ valid distance labels

$d = (3, 1, 2, 0)$ exact distance labels

2) (i,j) in $G(x)$ is <u>admissible</u>

if $d(i) = d(j) + 1$ (inadmissible otherwise)

3) <u>a path P from s to t in $G(x)$</u>

(i.e. an augmenting path) is <u>admissible</u>

if all its arcs are admissible

<u>Property</u> : an admissible path is a shortest augmenting path

<u>Proof</u>

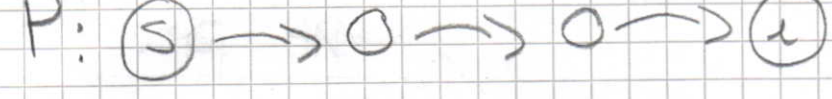
if the admissible path has K arcs, then $d(s) = K$, but $d(s)$ is a lower bound on the minimum cardinality of the paths from s to t in $G(x)$ ....

therefore K is such a minimum cardinality !

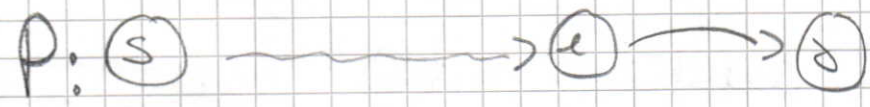
□

Shortest augmenting path algorithm

- The algorithm maintains a partial admissible path from s:



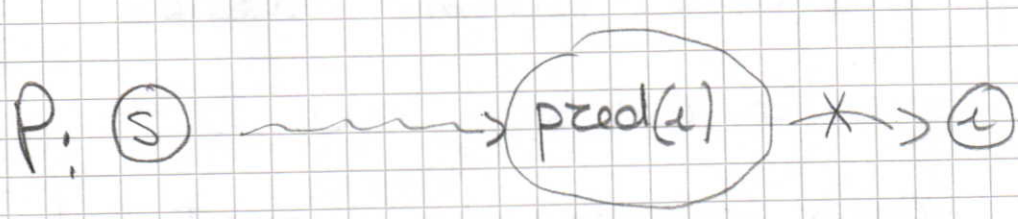
- Performs advance or retreat at i:



if $d(i) = d(j) + 1$ for an admissible arc: advance

(j current node)

otherwise (i.e. $d(i) < d(j) + 1 \ \forall (i,j) \in G(x)$)



retreat

$d(i) := \min\{d(j) + 1 : (i,j) \in G(x)\}$

refine d(i)

When the current node is t :

P is shortest augmenting path !

< push $\delta = \min\{r_{ij} : (i,j) \in P\}$ along P >

Time complexity : $O(n^2 m)$

no proof

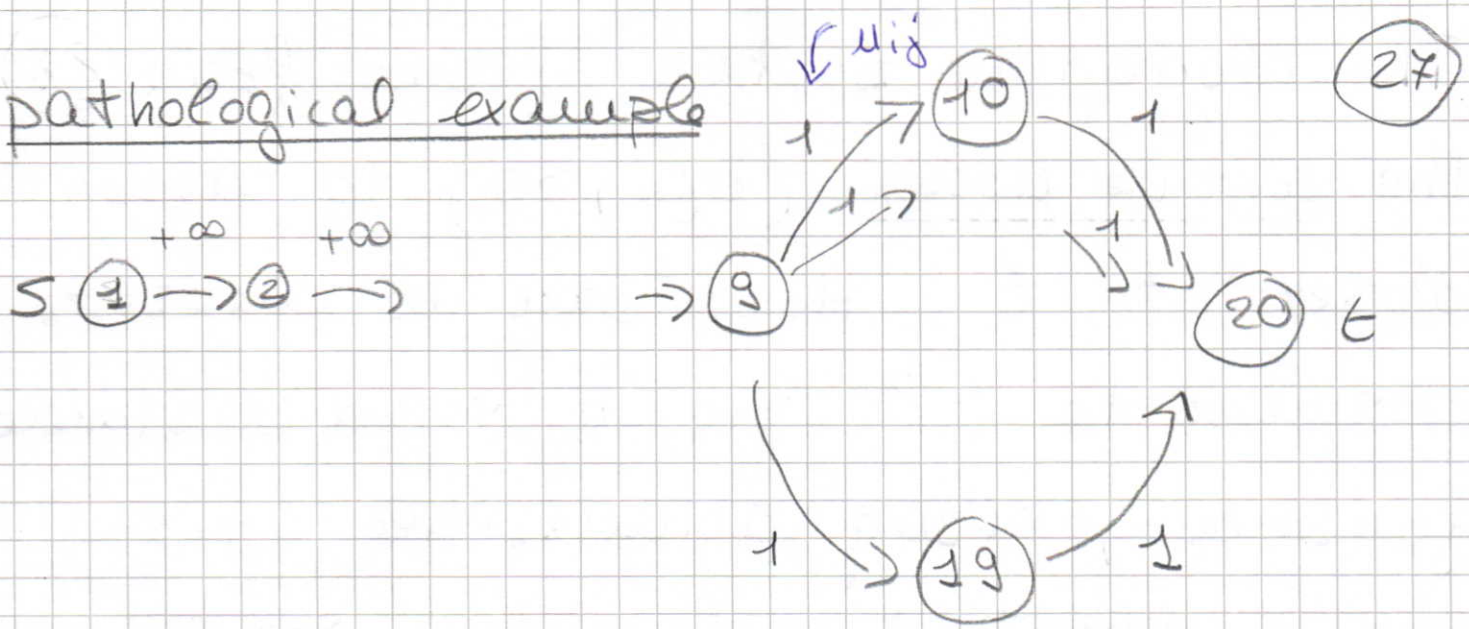
The preflow-push algorithm

- more powerful and flexible
- augmenting path algorithm requires $O(n)$ for each pushing flow operation (it may be expensive) ; more efficiently, preflow-push pushes flow along single arcs , by relaxing the flow conservation constraints

↓

preflow

pathological example



The (shortest) augmenting path algorithm pushes 1 unit of flow along 10 paths of cardinality 10! can we be more efficient?

The preflow-push algorithm maintains a <u>preflow</u> at each step, i.e. it relaxes the flow conservation constraints into:

$$\sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} \geq 0 \quad \forall i \in N \setminus \{s,t\}$$

Let $e(i) = \sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij}$

<u>excess of i</u>

The algorithm pushes flow from the <u>active nodes</u> ($e(i)>0$) to nodes "closer" to t. How can we measure closeness to t? Via a valid distance function, along admissible arcs (as in shortest augmenting path, but 1 arc at a time!)

Let $i : e(i)>0$

<u>Push / relabel (i)</u>

i.e. $d(i) = d(j) + 1$

<u>if</u> there is an admissible arc (i,j)

<u>then</u> push $\delta = \min\{e(i), r_{ij}\}$ along (i,j) — PUSH

<u>else</u> $d(i) := \min\{d(j)+1 : (i,j)$ is in $G(x)\}$ — RELABEL

if $\delta = r_{ij}$ the push is <u>saturating</u>;

<u>nonsaturating</u> otherwise

Initialization

$x := 0$;

$x_{sj} := u_{sj}$ $\quad \forall (s,j) \in FS(s)$;

< compute the exact distance labels $d(i)$, $\forall i$ > ;

$\boxed{d(s) := n}$

Observations

1) each i adjacent to s has a positive excess at the beginning (the maximum possible!)

2) $d(s) = n \rightarrow$ no augmenting path (due to 1)

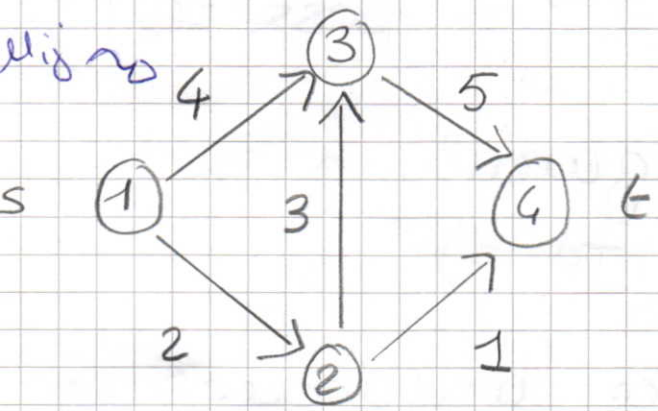
3) since all $d(i)$ are nondecreasing (see RELABEL), during the algorithm execution there is (no) augmenting path (no further flow is pushed from s!)

Termination : $e(i) = 0 \quad \forall i \in N \setminus \{s, t\}$

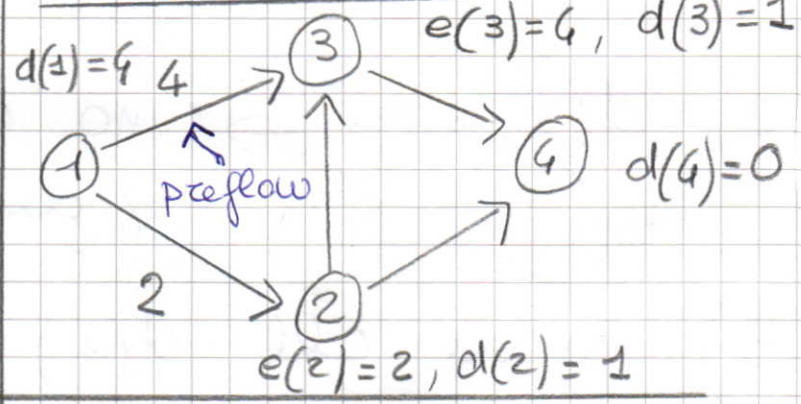
When this happens, the preflow has been converted into a flow, which is maximum since there is no augmenting path (from 3))

Algorithm correctness

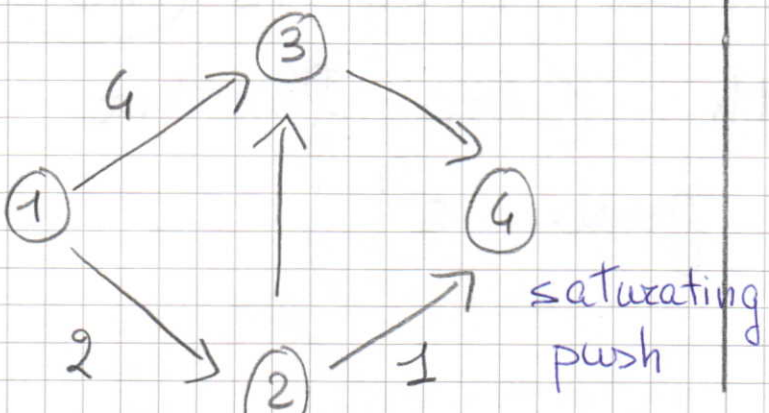
Example (partial, see page 227)



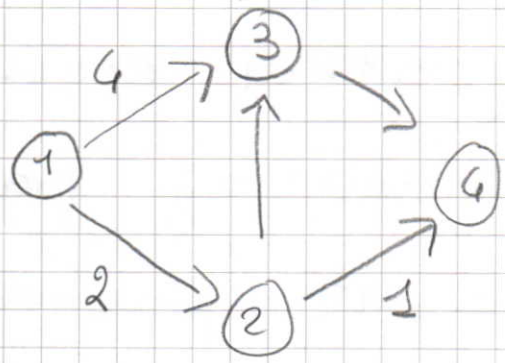
Initialization



Select 2, push on (2, 4)



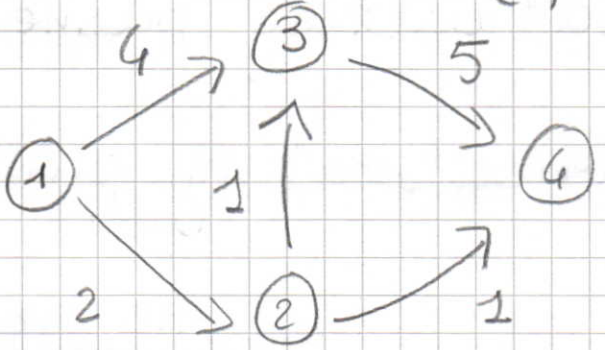
Select 2, relabel at 2



$d(2) := \min\{d(3) + 1, d(1) + 1\} = 2$

final snapshot

$e(3)=0$, $d(3)=1$



$d(4)=0$

maximum flow value 6

$e(2)=0$

$d(2)=2$

Minimum cut ?

$(\{\overset{s}{1}\}, \{2, 3, \overset{t}{4}\})$

but also

$(\{1, 2, 3\}, \{4\})$

s t

[If $u_{13} = 6$?]

Time complexity : $O(n^2 m)$

- $O(n^2)$ relabel
- $O(n \cdot m)$ saturating pushes
- $O(n^2 \cdot m)$ nonsaturating pushes

no proof

Specific implementations : depend on specific active node selection rules :

- FIFO order : $O(n^3)$
- Highest-label : $O(n^2 \sqrt{m})$

(no excess scaling rule)